DETERMINING BEHAVIOR OF BID-ASK PRICES FOLLOWING LIQUIDITY SHOCKS

JIA-HAN CHIAM, CHUANQI SHEN

1. Abstract

One significant challenge in the field of algorithmic trading is predicting the short-term behaviour of stock prices after market events. This project aims to predict the short-term behaviour of the market following a liquidity shock.

2. Introduction

Every stock has an "order book" of outstanding buy and sell orders, from which a highest bid price and lowest ask price can be derived. The latter is always higher than the lower (if someone bids higher than the lowest ask, the trade is immediately satisfied), and the difference is called the bid-ask spread. If a large trade occurs, all of the buy orders at the highest bid or sell orders at the lowest ask will be satisfied, and the bid price will fall or the ask price will rise accordingly, causing the spread to temporarily increase. This phenomenon is known as a liquidity shock, and the market will respond to it by refilling the order book. The new bid and ask prices may revert to their original levels, or there may be a permanent change in the price of the stock.

In this project, we use Machine Learning techniques to construct a model that can accurately predict a stock's subsequent bid and ask prices after a liquidity shock occurs.

3. Data Collection

This project is based on a Kaggle challenge [1], and uses data provided by the challenge organizer. The data set consists of over 700,000 training examples. Approximately 50% of the examples correspond to buyer-initiated liquidity shocks while the rest are seller-initiated. A single example is described by 7 auxiliary values (e.g. trade volume) and 100 trade or quote events in chronological order. The first 50 events (called predictors) are described by transaction time, type of transaction (trade or quote), bid price and ask price. A liquidity shock occurs between the 49^{th} and 50^{th} event. The next 50 events (called responses) are simply described by a bid and an ask price. Figure 1 illustrates the data provided:

Given the auxiliary data and the 50 predictors, our task is to predict the values of the responses. In order words, we want to predict how the bid and ask prices will change in the immediate aftermath of a liquidity shock. While we are provided with data for 50 response events, our project focuses on predicting only the first 10 responses.

Table 1 Data Schema

Variable Type	Predictors												Responses												
Event Time								t=1				...	t=49				t=50				t=51		...	t=100	
Column Id	1	2	3	4	5	6	7	8	9	10	11	...	200	201	202	203	204	205	206	207	208	209	...	305	306
Variable Name	row_id	security_id	p_tcount	p_value	trade_vwap	trade_volume	initiator	transtype<t>	time<t>	bid<t>	ask<t>	⋮	transtype<t>	time<t>	bid<t>	ask<t>	transtype<t>	time<t>	bid<t>	ask<t>	bid<t>	ask<t>	⋮	bid<t>	ask<t>

Trade Quote

Liquidity Shock

FIGURE 1. taken from http://www.kaggle.com/c/AlgorithmicTradingChallenge/data

4. Problem Formalization

The training examples are separated into two sets, corresponding to buyer-initiated and seller-initiated shocks. The two sets are analyzed separately. We randomly shuffled both sets and extracted 5% of the examples to use as our test set, while the remaining 95% were used to train our models. Our feature vector $x^{(i)}$ consists of 104 values: 4 auxiliary values that we felt were relevant (size of trade causing the liquidity shock, price of trade causing the liquidity shock, trade volume (in pounds) and trade volume (in number of trades)) and the bid and ask prices of the first 50 (predictor) events. We opted not to include the timestamps or transaction types in our feature vector. Our models output a vector $y^{(i)}$ consisting of 20 values, namely the bid and ask prices of the 10 (response) events immediately following the liquidity shock.

Given the training set (X_{train}, Y_{train}), we want to learn a hypothesis function $h(x)$ that can accurately predict $y^{(i)}_{test}$ given $x^{(i)}_{test}$. We used the error metric stipulated by Kaggle, evaluating a given $h(x)$ by calculating the root mean square of $||y - h(x)||$ for all pairs $(x, y) \in (X_{test}, Y_{test})$.

5. Methods

Our models are benchmarked against a naive algorithm, which simply outputs 10 copies of the last bid-ask price in the set of predictors. We devised three different algorithms for making predictions on the data. The first employs a simple linear regression model, and we used the normal equations to derive a θ vector that minimises the root mean square error of $h(x) = \theta^T(x)$.

The next two algorithms follow a three-phase framework we call Clustering-Classification-Regression (CCR):

5.1. **Clustering.** We divided the training data into clusters according to the general trend of the post-shock prices. For example, a stock may see its price revert to its original value after a liquidity shock, or the price may continue increasing or decreasing in the same direction. We accomplished this by running the K-means clustering algorithm on the 20-dimensional response vectors. We subtracted the last bid price of the predictor vector from each y and normalized the y vectors before running K-means clustering, so

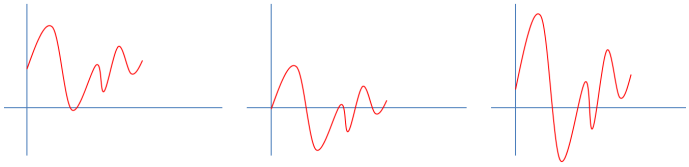

FIGURE 2. Left graph is off by translation factor, right graph is stretched

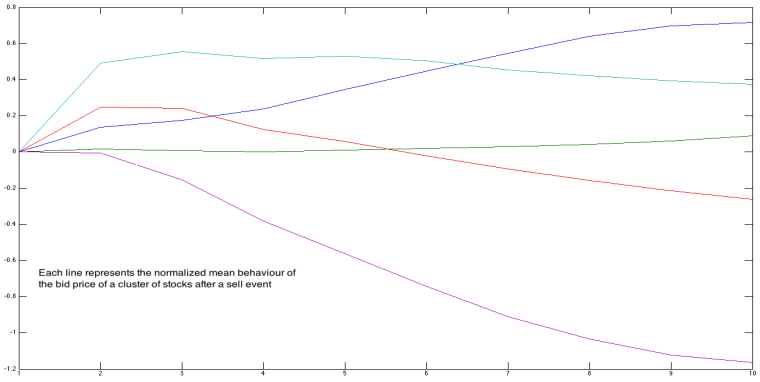

FIGURE 3. Graph showing the mean y vector for each cluster.

that graphs that are off by a translation or a scalar factor, as shown in Figure 2, will be categorised together. Our intention was to categorize stocks according to movement trends, not the absolute value of prices or prices changes. Figure 3 shows how the post-shock movement trends are broadly categorised.

5.2. **Classification.** After clustering the y vectors, we trained a classification algorithm to predict which y-cluster a data point (x, y) would belong to given the value of x. The first classification algorithm we employed was Gaussian discriminant analysis. The second was a multi-class support vector machine using an RBF kernel. We used the open-source tool LIBSVM[2] to perform the latter classification. We trained the GDA classifier on the entire training set, but we were only able to train the SVM classifier on $50,000$ examples as it was too slow to be applied over a larger data set.

5.3. **Regression.** We divided the training set into groups according to the cluster labels predicted by the classifier, and trained five separate linear regressions on each of these subsets of the training set, obtaining $\{\theta_1, \cdots, \theta_5\}$. When performing predictions on a test example $(x^{(i)}, y^{(i)})$, we first ran the classifier on $x^{(i)}$ to predict which cluster $c^{(i)}$ the $y^{(i)}$ vector would fall into, and then applied the corresponding linear regression model to generate a prediction $h(x^{(i)}) = \theta_{c^{(i)}}^T x^{(i)}$.

6. Results and Discussion

From Figure 4, we observe that the largest clusters in the buy and sell training sets only account for 30.3% and 25.7% of the data respectively. If the predictor and response

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
Buy	30.3%	20.4%	16.8%	15.6%	17.0%
Sell	15.5%	25.7%	23.0%	20.0%	15.9%

FIGURE 4. Distribution of training examples among the clusters.

	SVM	GDA
Buy	49.8%	34.4%
Sell	40.9%	28.9%
Overall	**45.3%**	**31.57%**

FIGURE 5. Classification accuracy of SVM and GDA

	Training Set Error	Test Set Error
Naïve	0.6192	0.6177
LinearReg	0.5793	0.5636
K-Means + GDA	0.5769	0.5613
K-Means + SVM	0.5716	0.5592

FIGURE 6. Comparison of training set and testing set error

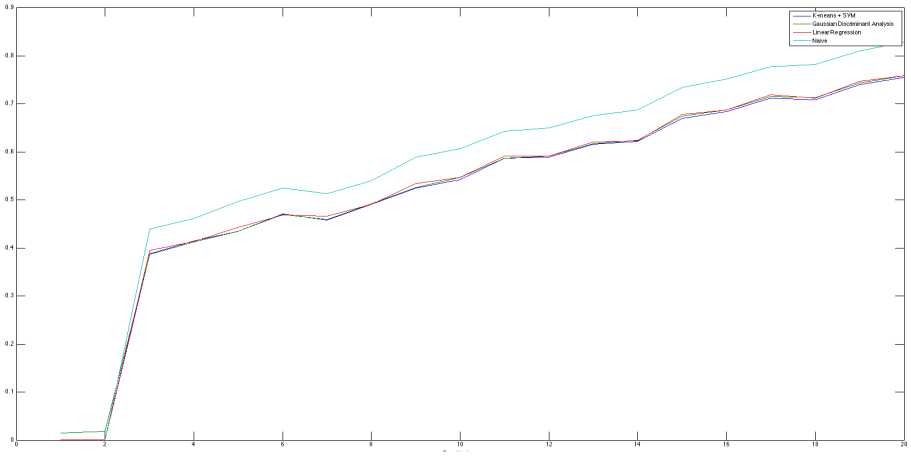

FIGURE 7. graph comparing testing set error between algorithms

vectors were completely independent and the test set was drawn from the same distribution as the training set (a reasonable assumption as the test set was randomly selected from the original Kaggle data set), it would not be possible to achieve accuracies higher than those values. Figure 5 shows that our SVM classifier in fact managed to achieve a classification accuracy of 45.3% over the whole training set. While GDA performed significantly worse, its accuracy is still high enough to offer some predictive value. This means that our classification algorithms were successful in detecting correlations between the pre-shock and post-shock bid-ask prices.

From Figure 6, we note that SVM and GDA performed better than a simple linear regression, with SVM producing the lowest test error. However, both CCR algorithms only performed slightly better than the original linear regression. Although our classification algorithms are able to uncover patterns in the pre-shock and post-shock bid-ask prices, our linear regression model was unable to successfully exploit these patterns.

Due to the large size of our training set, overfitting was unlikely to be an issue, and indeed Figure 6 shows that our test set error was in fact marginally lower than our training set error. Our models are therefore suffering from high bias, rather than high variance.

7. Future Directions and Improvements

The high bias means that we may be able to improve on the algorithm if we use higher-dimensional feature vectors, or perform a support vector regression with a nonlinear kernel.

We can also consider changing the error metric used to evaluate the algorithms, and the cost function the algorithms seek to minimize, as the root mean square error may not accurately reflect the predictive power of our models. For example, if we were able to estimate a confidence level for each prediction (e.g. using the classifier's confidence in the label prediction), and identify certain predictions as being more likely to be correct while others were close to random guesses, we would be able to determine which predictions we could build profitable strategies around and which we could ignore. Such a model would be superior to one which gave the same root mean square error over the whole test set, but which did not indicate each prediction's confidence level (e.g. simple linear regression).

The root mean square metric also heavily penalises large deviations, and the algorithm is hence incentivised to produce conservative values, even if it has correctly identified a trend in the data. For example, if a model determines that a price has a 60% chance of increasing from y to $y + \epsilon$, and a 40% chance of decreasing to $y - \epsilon$, outputting a value of $y + \epsilon$ would result in an expected mean square error of $0.4 * (2\epsilon)^2 = 1.6$, whereas the naive algorithm which outputs y would produce an mean square error of 1, even though the latter has no predictive power whereas the knowledge that a price is 1.5 times more likely to increase than decrease would be considered valuable information. If the algorithm were trying to minimise the mean square error, it would output $y + 0.2\epsilon$, for a mean square error of 0.96 (only very slightly better than the naive algorithm, even though the model has significantly more predictive power). This value of $y + 0.2\epsilon$ is arguably less useful than a prediction of $y + \epsilon$, and certainly less useful than the full representation of the model's internal state (a 60% chance of the price being $y + \epsilon$ and a 40% chance of being $y - \epsilon$), which the root mean square metric has no way of evaluating.

References

[1] Algorithmic trading challenge - kaggle. http://www.kaggle.com/c/AlgorithmicTradingChallenge.

[2] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.